

# A Context-based System for Personal File Retrieval

Hung Ba Ngo<sup>1, 2</sup>

College of Information Technology & Communication  
Cantho University<sup>1</sup>  
01 Ly Tu Trong, Cantho, Vietnam  
nbhung@cit.ctu.edu.vn

Frédérique Silber- Chaussumier, Christian Bac

Department of Informatics  
TELECOM & Management SudParis (ex INT)<sup>2</sup>  
9 Charles Fourier, 91011 Evry Cedex, France  
Frederique.silber-chaussumier, Christian.bac@it-sudparis.eu

**Abstract**— This paper introduces CoFS, a context-based system for personal file retrieval. Users in this system use tags to describe files or resources of special interest. A set of tags assigned to a file by a user is called a tag-based context. For each user, his interesting files are organized into the appropriate contexts. A directed acyclic graph of tags is created for each user to help him navigate from one context to another to retrieve his files.

*Semantics; file system; tag; context; file; retrieval; personal*

## I. INTRODUCTION

Today there are an ever-increasing number of files on our desktops. Furthermore, files of special interest to the user, so-called *personal files*, are now stored not only in local disks but also on other sources such as company file servers, the Internet or mail boxes. Many tools are used to access such personal files; a file manager, a web browser, a user mail agent,... These tools have different methods for file searching, as directory browsing, hyper-link browsing, or scanning a list of mail headers. This tool diversity associated with the increasing number of resources makes it difficult for a user to revisit his personal files. Although traditional file systems support a *link* mechanism, the file system is not used as the central point to access personal files. One must provide a single method to access such resources. We propose CoFS, a Context-based System for personal File retrieval, to address this issue. CoFS uses a personal tag organization presented in [1]. Tags are organised into tag-based *contexts*. A context in our approach is *a set of tags assigned to a resource by a user*. As a matter of fact, tagging systems are widely used to tag personal resources: on the Internet, in desktop search tools and also in some semantic file systems. To retrieve personal resources from contexts, we adopt our DAGoT algorithm [1] that classifies files into appropriate contexts and create a hierarchy of contexts in which users can navigate. Implementing this approach CoFS captures contexts from user tagging process. In section 2 of this paper, we firstly present the general tagging system model and then explain the differences between existing tag-based searching model and our context-based model. A brief description of the algorithm DAGoT [1] is presented in section 3 and 4. At last, we present the implementation of the COFS prototype in section 5. The last section contains conclusions and future works.

## II. DEFINITION OF A TAG-BASED CONTEXT

Nowadays, tagging systems such as *Delicious* [2], *Flickr* [3] are widely used on the Internet. These tagging systems enable users to add *tags* (or keywords) to Internet resources without relying on a controlled vocabulary. On desktops, tags are also used for personal file retrieval. Users of *Spotlight* [4] can assign a tag to a set of semantically related files. In the file system domain, *LFS* [5] allows users to associate files with tags representing file properties. In this paper, we consider the notion of *resource* representing files but also different other kind of resources as URL,...

In a tagging system, a user makes a *post* to assign a file with a set of tags. Each tag represents a concept or an object related to the file. In addition, the set of tags assigned to a file represents a topic, or a subject relevant for the file owner. For example, a user may assign the set of tags {*Vacation, Paris, 2007*} to the file *myphoto.jpg* to recall that the photos were taken during the vacation of summer '07 in Paris. Thus, a naïve tag-based search will return all the files that hit a given tag or a logical expression (and, or, not) of tags. Tags give flexibility to the users in describing their personal files for later retrieval. Nevertheless it does not classify search results and does not support any browsing method. Too many returned files in the search result without classification greatly reduce the relevancy of a tagging system. In addition, without browsing, a user has no help to choose relevant tags that should be included in queries. Much research addresses these issues. In the system *Delicious*, two or more tags associated to the same resource are considered as *related*. A list of resources tagged to a chosen tag and a list of its related tags are returned. Related tags are a means of navigating between tags to revisit sought out resources. However, when the number of resources and tags increase, parsing the result or choosing a right tag from related tags to refine a searching result becomes a tremendously complex task for a user.

In file systems, *LFS* [5] allows users to associate files with tags representing file properties. *LFS* supports axioms between tags like a parent-child relationship. Users can *manually* create axioms between their tags. A taxonomy of tags is created based on axioms. Users can navigate in the taxonomy for file retrieval as they do with traditional directories. *TagFS* [6] organizes tags into related tags as in *Delicious*. Consequently, *TagFS* encounters the same problems when the number of files and tags increases.

Our approach differs from the above mentioned approaches in that it introduces the notion of *context* into a tagging system and provides *context-based searching* instead of tag-based searching. A *context* (or a tag-based context) is a set of tags assigned to a file by a user. As a matter of fact, a user can better understand the meaning of a tag when it is placed in a context. For example, the meaning of *JDIR* is clearer if we place it in the context  $\{Article, JDIR, 2007\}$ . Usually, a tag belongs to many contexts. Our DAGoT algorithm [1] classifies the contexts into a hierarchical structure from general to specific contexts. Instead of returning all files that hit a tag, a context-based searching only returns the files that hit the *most popular context* of the tag.

Table I is an example of posts made by a user. If the user did a tag-based search with the tag *Article*, five files (from *file1* to *file5*) would be returned. These files belong to three contexts  $\{Article, JDIR, 2007\}$ ,  $\{Article, JDIR, 2007, FinalVersion\}$ , and  $\{Article, 2007, RIVF\}$  that are classified by the DAGoT algorithm [1]. This classification is represented in the “Fig. 1”. Instead of tag-based searching, a context-based search launched for the tag *Article* will only return *file1* and *file2* in the most popular context  $\{Article, JDIR, 2007\}$  of the *Article* tag. If the user is not satisfied, he can navigate down to search a relevant resource in the other specific contexts of the *Article* tag. In the next section we will introduce the DAGoT algorithm.

### III. CREATING A DIRECTED ACYCLIC GRAPH OF TAGS

The CoFS prototype we present in this article uses the Directed-Acyclic Graph of Tags (DAGoT) presented in [1]. Inside the CoFS prototype, several graphs are built: one graph organizes the personal resources of one user. An important point in a tagging system as CoFS, is that tags give flexibility to the user to describe personal resources for later retrieval. Thus we suppose that each user possesses a personal vocabulary created from his own tags. For each tag of a DAGoT, the files that hit the most popular context of this tag, are given. In addition the children tags and parent tags guide the user to the less specific contexts and the more general contexts respectively.

TABLE I. EXAMPLE OF POSTS MADE BY A USER

Files	Tags assigned to files
file1.doc	{ Article, JDIR, 2007 }
file2.pdf	{ Article, JDIR, 2007 }
file3.doc	{ Article, JDIR, 2007, FinalVersion }
file4.pdf	{ Article, JDIR, 2007, FinalVersion }
file5.doc	{ Article, 2007, RIVF }
file6.pdf	{ 2007, FinalVersion, RIVF }
file7.jpg	{ 2007, Vacation, Paris }
file8.jpg	{ 2007, Vacation, Hanoi }

A DAGoT is composed of the following basic concepts:

- Related tags: two tags assigned to the same file are considered as related with each other.
- Popularity of tag: the popularity of a tag is the number of files assigned to the tag.

- Upper tags: supposing that  $t1$  and  $t2$  are two related tags with the popularities respectively  $p1$  and  $p2$ .  $t1$  is called an upper tag of  $t2$  if  $p1$  is greater than  $p2$  or if  $p1$  is equal to  $p2$  then the label of  $t1$  should be smaller than the label of  $t2$ .

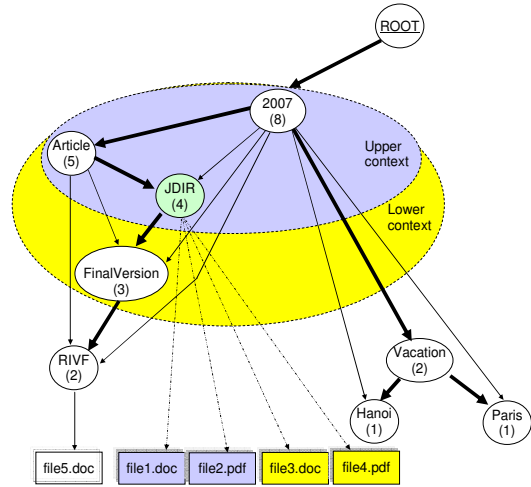


Figure 1. DAGoT created from the posts in the table I

Creating a DAGoT consists in the following steps. First, the popularity of each tag is calculated. Next the upper tags are calculated for each tag. “Fig. 2” presents the popularities and the upper tags of each tag in Table I. The third step calculates the parents for each tag. For each tag, the least popular tag among its upper tags will be accepted as its parent tag. “Fig. 1” represents a DAGoT created from the posts in Table I.

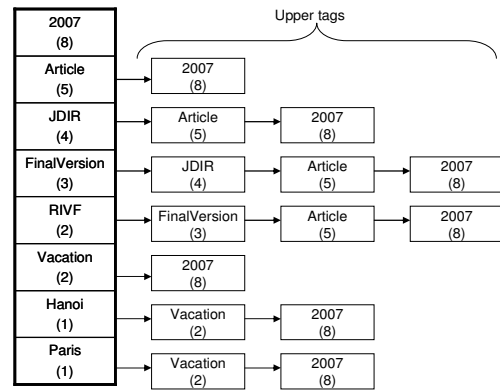


Figure 2. Popularity and upper tags of tags in Table I

From this point if the user makes new posts as described in Table II, then the popularities of the existing tags and their upper tags are modified. In our example, the new popularity of the existing *Article* tag and the popularities of the new tags are computed in “Fig. 3”. In fact, the computation of parent tags can be more complex depending on the upper tags of the given tag. When upper tags of a given tag do not belong to one single context there will be several parent tags. Each parent is the least popular tag of each upper context. *Conference* is an example for this case. *Conference* has three upper tags *2008*, *Article* and *Trégestal* that belong to two different contexts

{Conference, 2008, Article} and {Conference, 2007, Trégestal}. The least popular tag in each of these contexts will be a parent of the Conference tag. Therefore Conference has two parents Article and Trégestal. The new part of the DAGoT corresponding to the posts in Table III is presented in “Fig. 3”.

TABLE II. NEW POSTS ADDED BY THE USER

Files	Tags assigned to files
file9.pdf	{Conference, CORIA, 2008, Trégestal}
file10.doc	{Conference, CORIA, 2008, Article}
file11.jpg	{Trégestal, 2008, Photo}
file12.jpg	{Trégestal, 2008, Photo}
file13.jpg	{Trégestal, 2008, Photo}
file14.jpg	{Trégestal, 2008, Photo}

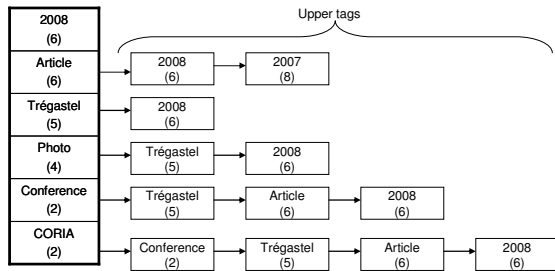


Figure 3. Popularity and upper tags of tags in Table II

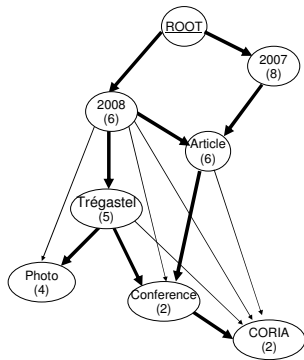


Figure 4. A new part of DAGoT created after adding new posts in Table II

#### IV. USING DAGoT FOR CONTEXT-BASED SEARCHING

The previous section shows how to create a DAGoT from the posts made by a user. This section presents how to use a DAGoT to provide a user with a context-based searching. We use classical Unix file commands to illustrate context-based searching in CoFS.

When a request is made, corresponding to the “cd” command, the current working directory is replaced by the most popular context of the given tag. It becomes the *current* context. While the “cd tag” command is used to change the current context, the “ls” command is used to search resources in the current context. “Fig. 5” shows some examples of context-based searching. The command (1) changes the current context to {2007, Article, JDIR}. A request (2) in this current context returns the *satisfying resources*. Such resources are tagged only with the tags included in the current context. In our

example, the *file1* and *file2* resources tagged with all the {2007, Article, JDIR} tags will be returned as satisfying resources. In addition, a list of parent and children tags of the current context are also returned as a result of the search. Users can choose a child tag to navigate to a more specific context (3), (4), or choose a parent tag to navigate to a more popular context. In fact, the satisfying resources are not always returned for every tag. The commands (5) and (6) illustrate the case where a tag has more than one most popular context. In this example *Vacation* participates in two different contexts {*Vacation, 2007, Paris*} and {*Vacation, 2007, Hanoi*}. The children tags *Paris* and *Hanoi* of the command (6) help the user choose between one of these two contexts.

```

{/}cd Article (1)
{/2007/Article/JDIR}ls (2)
file1.doc file2.pdf
Children: FinalVersion
Parent: Root
{/2007/Article/JDIR} cd FinalVersion (3)
{/2007/Article/JDIR/FinalVersion}ls (4)
file3.doc file4.pdf
Children: RIVF
Parent: JDIR
{/2007/Article/JDIR/FinalVersion}cd Vacation (5)
{/2007/Vacation}ls (6)
Children: Hanoi Paris
Parent: 2007

```

Figure 5. Example of the use of a DAGoT for context-based searching

#### V. PROTOTYPE FOR COFS

In this section, we introduce *CoFS*, a prototype for a *context-based system for personal file retrieval*. CoFS allows a user to flexibly describe their personal files located in different sources via tags and then apply the DAGoT algorithm to provide an efficient context-based searching for personal file retrieval. Files located in home directories are also indexed by CoFS to provide a more advanced search combining context-based and text-based searching.

The CoFS functions are grouped into modules as “Fig. 6” describes. As a *file event* e.g. *file created, content modified, file renamed, or file deleted* is fired, the *EventManager* analyses the configuration files to determine the modules that should be activated. If a *file created* event is sent to *TagCollector*, a *Tagging Form* in Fig. 7 will be displayed to allow the user to assign a set of tags to the corresponding file. Information about this post is sent to *DAGoTManager* to update the DAGoT for the current user. If the type of this file has already been registered to be indexed, the event is also sent to the *IndexManger* for updating the index file of the current user.

The *SearchManager* has a user interface as shown in Fig. 8 It allows a user to do a context-based search, or a conjunction of a context-based search with a text-based search. The user can handle satisfying resources returned by the *SearchManager*, such as modifying tags assigned to one file or many files, deleting one ore more files, executing or opening a file.

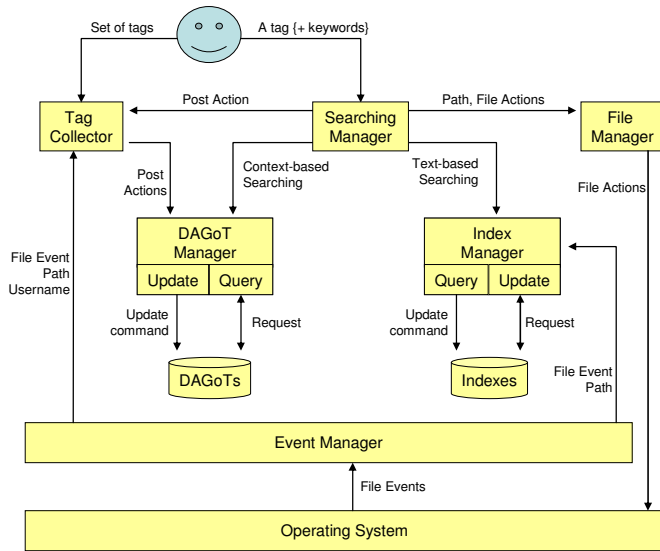


Figure 6. Function model of CoFS

To be able to extend the system with multiple types of personal resources, we adopt a plugin structure to implement CoFS. We choose the Java Plugin Framework JPF [7] to build CoFS. The architecture of CoFS is described in “Fig. 9”. The heart of CoFS system is the *plugin-core* that implements the function of EventManager module in “Fig. 6”. *Plugin-core* uses *Jnotify* [8] to capture file events fired from the kernel. *Plugin-core* defines the interface *UserListener* to receive the actions that a user wants to make on satisfying resources returned in a search result such as opening or executing a resource; deleting one or more resources; changing tags of one or more resources. File events and user actions will be sent to the class *Actioner* for analyzing. From there, the class *Actioner* will call some plugins to reply to the file event (or the user action). To quickly obtain a prototype for CoFS, we use the inference engine *Flora-2* [9] to implement and maintain the DAGoT for each user. The indexing library *Lucene API* [10] is used by the *IndexManager* plugin to index document contents. To maximize the autonomy of the plugins, the communication between two plugins or between a plugin and an API are all made via our own XML format. The functions of the *FileManager* module in “Fig. 6” are distributed into two plugins: the *StoreManager* and the *ExecutionManager* plugins. In the current version, we implemented the *StoreManager* plugin that operates on the Linux virtual file system. The *ExecutionManager* plugin is responsible for executing or opening different types of resources.

The current version of CoFS allows a user to assign tags not only to files on local disk but also to resources on the Internet. An extension plugin, called *CoFS-Tag*, is developed for the Firefox browser. A *Tag* button will appear on the Firefox toolbar when the extension plugin is installed. Pressing

on this button, a dialog will be displayed to assign a set of tags to the current URL in the Firefox browser. The plugin will create a *reference file*. The name of such a reference file is created from the creation time of the file and the file extension *ref*. This file contains two lines: the first one is the URL of the current web page in the browser; the second one is the list of tags assigned to the resource. When receiving a creation event of a reference file, the *Actioner* analyses the content of the reference file and then forms a “mute tagging” request message to send to the *TagCollector* plugin. A “mute tagging” request message contains a resource and a set of tags to be assigned to the resource, therefore the *TagCollector* plugin does not need to active the Tagging form shown in “Fig. 7”, to collect a set of tags from the user.

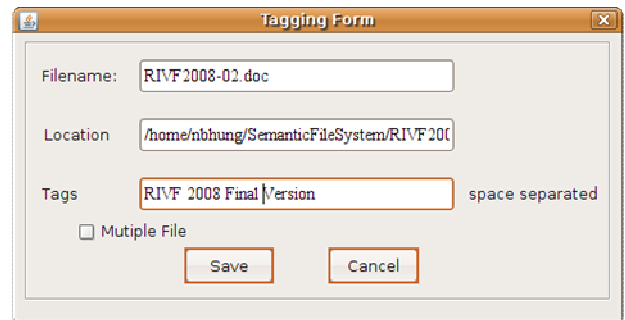


Figure 7. The user interface for tagging actions in CoFS

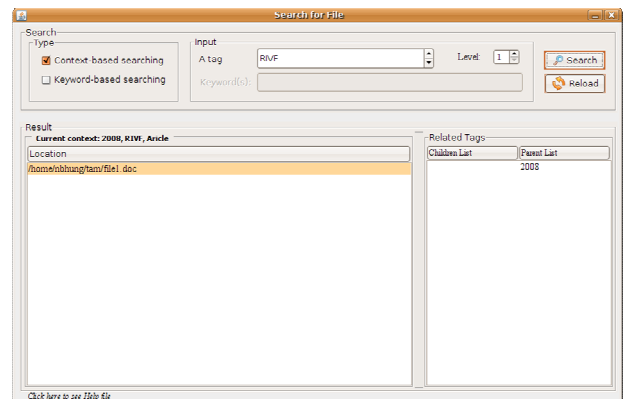


Figure 8. The user interface for personal file retrieval in CoFS

“Mute tagging” is also used in the case where a user wants to assign many files with the same set of tags, such as importing photo files from a camera. If the option *Multi files* in the Tagging form, “Fig. 7” is checked, the set of tags in the the Tagging form is used not only for the current file but also for the *successive files*. The interval time between the creations of two successive files has to be smaller than a *predefined constant*. Our experiment proposes 1000 ms for this interval.

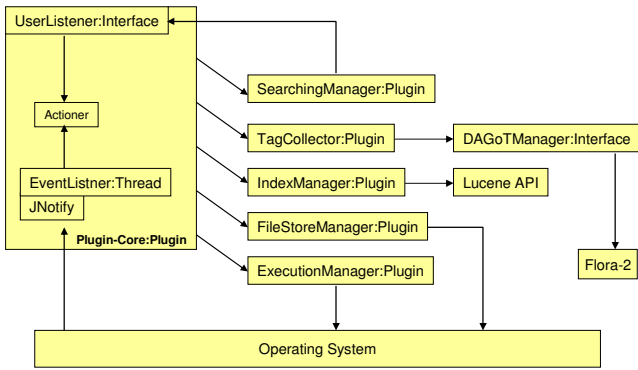


Figure 9. Architecture of CoFS

## VI. CONCLUSION AND FUTURE WORKS

Resources of special interest that a user will revisit in the future are called *personal files*. A user has to use many tools with different searching methods to retrieve his personal files that are located in different sources. In this paper, we have presented our CoFS prototype. CoFS allows users to assign tags not only to files on local disks but also to resources on the Internet. A set of tags assigned to a resource by a user is called a *context*. Using the DAGoT algorithm [1], personal files are organized into appropriate contexts. The algorithm also classifies such contexts into a hierarchical structure based on the popularity of tags. A directed acyclic graph of tags is created for each user. CoFS is proposed as a central point to provide access to personal files located in different sources with a unique *context-based searching* method. Navigating in a DAGoT, a set of resources satisfying the most popular context of the current tag will be returned. In addition, choosing a parent tag or a children tag will guide the user to a more popular or less specific context respectively. In CoFS,

documents can be indexed to provide users with an advanced searching method which is a conjunction of context-based searching and text-based searching. For future works, CoFS needs to be enhanced with two important aspects. First, because the current version of CoFS uses Flora-2 to update and maintain the DAGoT for every user, the response time is too slow. We need another implementation of DAGoT that is more interactive. Second, CoFS uses a *pathname* to refer to the resource content stored on local disks.. It means that the name of a directory may appear in many places in a DAGoT. If the name of the directory is changed by the underlying file system, it should be updated in the DAGoT too. This task costs too much. So we need another method to refer to the resource content to should replace the pathname.

## REFERENCES

- [1] Ngo, H.B., Bac C., Silber-Chaussumier F., "Enhancing Personal File Retrieval in Semantic File Systems with Tag-Based Context", EGC'2008, Volumes I, pp. xx-xx, Revue des Nouvelles Technologies de l'Information - RNTI E-11, Cépaduès, 2008.
- [2] Delicious. <http://del.icio.us/>.
- [3] Flickr. <http://www.flickr.com/>.
- [4] Apple Computer, Inc: Tiger Developer Overview Series - Working with Spotlight, 2005, <http://developer.apple.com/macosx/spotlight.html>.
- [5] Padioleau Y., Ridoux O., "A Logic File System", *Proceeding of 2003 USENIX Annual Technical Conference*, General Track, pp. 99-112.
- [6] Bloehdorn S., Görlitz O., Schenk S., Völkel M., "TagFS --- Tag Semantics for Hierarchical File Systems". *Proceedings of the 6th International Conference on Knowledge Management (I-KNOW 06)*, Graz, Austria, September 2006.
- [7] Java Plugin Framework, <http://jpf.sourceforge.net>.
- [8] Jnotify, <http://jnotify.sourceforge.net>.
- [9] Flora-2, <http://flora.sourceforge.net>.
- [10] Lucene, <http://lucene.apache.org/>.